Reliable Broadcast, Membership and Diagnosis in Distributed Systems

by

Meenu Jain

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY KANPUR

FEBRUARY, 1995

Reliable Broadcast, Membership and Diagnosis in Distributed Systems

A Thesis Submitted in Partial Fulfilment of the Requirements for the Degree of
Master of Technology,
by

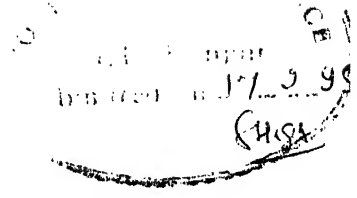
Meenu Jain

Department of Computer Science and Engineering
Indian Institute of Technology Kanpur

February 1995

CSE-1995-M
JAI-REL

CSE-1995-M-JAI-REL



CERTIFICATE

It is certified that the work contained in the thesis titled Reliable Broadcast, Membership and Diagnosis in Distributed Systems, by *Meenu Jain* has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

Pankaj Jalote.

February, 1995

Dr. Pankaj Jalote
Associate Professor,
Department of CSE
IIT, Kanpur.

Abstract

Distributed systems consist of a number of processors interconnected by communication links in a manner so as to enable various processors to communicate with each other. Frequently many distributed applications require that consistency of information be maintained among various processors. Reliable broadcast, membership, and diagnosis are three key problems in distributed systems that require consistency of information at different nodes in a distributed system. Several protocols for each of these properties have been developed earlier.

In the following work we have studied the various existing protocols for these properties and have described these properties at an abstract level such that they are protocol independent. These properties have been compared to see if they can be used interchangeably. We have shown that even though protocols for these properties have usually been developed in isolation, there are lots of similarities between these properties and in many situations protocols for one property can be extended easily to support another property. We have used the reliable broadcast property and established that membership and diagnosis can be achieved if reliable broadcast property is true in a given distributed system. Similarly it has been shown that reliable broadcast can be achieved given that any of membership or diagnosis is true. We have also done a comparison of the different protocols used for these properties, based on the number of messages used and the time taken by each protocol.

Acknowledgments

As I take a step further in my academic career, I wish to convey my heartfelt gratitude to several persons. First and foremost I wish to express my sincere gratitude to my thesis supervisor Dr. Pankaj Jalote for his able guidance and unfailing help, at every stage of my thesis. His constant support and professional approach has made it possible to achieve the results. Working under him has been a memorable experience.

My special thanks to S. Govindraj, Hari Babu and Haripriya for devoting time to go through my work despite their busy schedule. I thank my friends in the girl's hostel, who made my stay in IIT Kanpur a memorable one.

I take this opportunity to express my heartfelt gratitude to all members of my family, who have always stood by me. Their constant support, encouragement, love and complete trust in me has enabled me to work towards achieving success.

Lastly I wish to thank all those who have directly or indirectly helped me throughout my work.

Contents

Chapter 1

Introduction

1.1 Introduction to distributed systems

Distributed systems are defined as a number of autonomous computers, where all of them communicate via a network. The system is provided with distributed system software that enables the computers to coordinate their activities and utilize all the different resources efficiently. A well designed distributed system works as a single integrated computing facility, all of whose resources are available to the user, even though the resources may be at several different places.

Major characteristics of distributed systems include resource sharing, concurrence, transparency and fault tolerance.

- Resource sharing allows a large number of the system's resources to be usefully shared among different computers. These include various hardware resources and software services. The main advantage of resource sharing is the reduction in cost.

- Concurrence in a distributed system allows several processes to concurrently execute on a computer. There are several applications in distributed systems that require parallel execution of more than one process on a single computer hence using the concurrence property.

- By transparency it is meant that the system user is unaware of the fact that the system's resources are distributed. To a user, the system is a single integrated unit providing all resources and facilities.

- Fault tolerance deals with all aspects related to tolerance of the system against failures. Different types of failure may occur in the system, and fault tolerance against these

allows the system to minimize the effect of faults and recover from faults gracefully. Fault tolerance is achieved in different ways through hardware or through software. Hardware redundancy requires use of multiple(redundant) resources of same type, so as to detect and mask faults. Software recovery means, recovery through different software that are designed to recover from various faults.

Other features of distributed systems include openness, security aspects etc.. All these aspects of distributed systems make the design of distributed systems a challenging task. In the distributed systems, the software required to achieve efficiency and proper working of the underlying hardware should be developed keeping in mind all the above mentioned features.

As the size of the distributed system increases, so also do the number of faults. Providing fault tolerance feature in a distributed system is a major part in the development of distributed system software. There are several different issues related to fault tolerance in distributed systems. These include reliable delivery of application messages despite failures, consistency of user and system control data, diagnosing failures and recoveries in order to provide consistency etc.. The following work deals with the above mentioned issues related to fault tolerance in distributed systems.

1.2 Motivation for current work

With the emergence of distributed systems, the applications based on distributed systems have been constantly increasing. As more and more distributed applications are emerging it has also become necessary that the systems supporting such applications be fault tolerant. Many distributed applications require some consistency between the information at the different nodes. For fault tolerance the consistency needs to be preserved despite failures in the distributed system. Generally, protocols are used to ensure the desired consistency properties. Reliable broadcast, membership and diagnosis are all examples of the desired consistency properties.

A reliable broadcast protocol guarantees reliable distribution of messages to all the nodes in the system. That is if a node broadcasts a message then a copy of the message should reach all alive processors in the system within some fixed time delay after it has been sent. Message broadcast is a major requirement in all types of network systems, specially in distributed systems. The messages that are broadcast should reach all the nodes in the

network. But sometimes due to transmission errors or due to buffer space not being enough, the message may not reach all nodes. This causes inconsistencies in the data at different nodes of the system. This could cause different nodes to behave differently causing errors. Hence it becomes essential that the distributed systems support reliable broadcast, and hence several protocols for the same have been developed.

Membership information consists of a consistent system-wide information about which processors are alive and which have failed in a distributed system at a given time. This information is attained by the membership protocol. Membership for a group of processors changes when it's processors either fail, or when some failed processor or processors recover from failure, or when a new processor joins or leaves the group. In any such case it is essential that there is a consensus on the membership of the group among the members. In case there is no agreement among different processors, membership server for one processor may behave inconsistently with another membership server. In order to maintain consistency, agreement among membership information is essential.

The system diagnosis protocol allows the fault-free nodes to test other nodes and reach a consensus regarding the fault situation of the system based on the test results attained. Undetected failures or faulty nodes cause problems for distributed system and their applications. Detection of faults by other nodes in the system lowers the overhead for testing and detecting faults. Results of the tests should be known to all nodes for consistency.

In all the above methods, consistency of information is the main aim. Such a property is required to construct a dependable system where all processors have consistent information based on which they can take decisions. Protocols have been proposed to ensure these properties.

Mostly, these properties have been studied in isolation and little work has been done in employing protocols for one property to achieve another. This is highlighted by the fact that protocols for one property rarely refer to work done on protocols for other properties. In this report we show that there is a lot in common between these properties and the systems with fail-stop processors and no partitioning protocols to support one property can be used to support other properties. This implies that for connected systems with fail-stop nodes(as is generally assumed) these three consistency properties are the same at an abstract level, and further work on these should consider these properties together and as minor variations of one property.

1.3 Thesis Overview

The 2nd chapter consists of the description of the system model used, and the assumptions made. It also contains the abstract specifications of the reliable broadcast, membership and the diagnosis properties.

The various existing protocols for the given properties have been briefly described in the 3rd chapter. The complexity analysis in terms of number of messages needed and time taken to achieve the property for the described protocols is also contained. It also includes the table giving the comparative results.

Chapter 4 describes the relation between the three properties. It contains the algorithms and proofs used for conversion for different cases suggesting that the protocols for one property can be used to achieve the other.

Chapter 5 consists of the conversion examples to convert one of the existing protocol for each property to achieve another. The conversion method used is that which has been described in the previous chapter.

Chapter 6 contains the summary of the work done and the results obtained. It includes suggestions for future work that can be taken up.

Chapter 2

Reliable Broadcast, Membership and Diagnosis Properties

In this chapter we first define our system model, and then give the abstract property of each of these properties. These specifications form one basis of development of protocols for these properties. We will later use these specifications to show how a protocol satisfying the specifications of one can be used to satisfy another abstract property.

2.1 The System Model

We consider a distributed system consisting of N nodes connected by a communication network(the properties and assumptions made for the system follow). The processor hardware consists of CPU, I/O processors, storage etc. and the software consists of operating system and communication subnet. The communication subnet accepts messages from and delivers messages to processors, manages message queues and monitors and controls the transmission on physical network links. We assume that a reliable point to point communication service is available i.e. if the sender and receiver nodes have not failed then a node (sender) can send a message to another node (receiver) reliably. Any two pair of nodes can communicate with each other such that no error occurs during transmission.

No assumption about the network topology is made except that it stays connected. The network contains a number of paths between any two pairs of nodes (p, q) so that inspite of failures node p has at least one communication path to q. In other words all alive nodes can always communicate in a reliable manner despite failures of certain communication links.

Processors here are fail-stop[15] which means that they stop functioning on failure. This means that a failed node will not communicate with any other node, as long as it remains

failed. It will not transmit any erroneous message, and it will not perform any wrong function as changing correct messages etc.. Such a node will just stop performing any functions as if it does not exist in the system.

We assume that each network node can test some other network nodes even in presence of failures of some links and nodes, and identify the ones that are faulty. In a system with fail-stop nodes the "test" usually is a message requesting a response, and a node is diagnosed as failed if it does not respond within some time duration. We assume that each alive node is being tested by some other alive nodes. In other words, if a node fails, it's failure is detected by some other node in finite time. Note that in some protocols the "testing" of nodes by other nodes is made an explicit part of the protocol, while in others it is merely assumed that some nodes detect failure of a node. Both these are essentially different forms of the same assumption.

The system S is modeled as a triplet $(V(S), E(S), T(S))$, where $V(S) = (0, 1, ..., N-1)$ is the set of nodes or vertices of the system S where i denotes the i-th node of an N node distributed system. $E(S) = (...\{i,j\}...)$ is the set of edges of system S where $\{i,j\}$ represent an undirected edge between node i and node j. $T(S) = (...\{i,j\}...)$ is the set of tests of system S where $\{i,j\}$ is a directed edge representing a test, where for all nodes i and j, node i if alive can test node j for correctness. The values $(V(S), E(S), T(S))$ are assigned to all nodes at the time of initialization and hence system model given above describes the initial state of the system. This information remains same unless the initial configuration of the system has to be changed.

A message m sent by a node p is always received by another node q within a known network delay which is dependent on the network diameter(it is the largest end to end distance between any two network nodes) and worst case processing delays. We assume that an upper bound of Δ time units is specified on this delay. If a message is sent by a node at time T and is not received by the receiver nodes by time $T + \Delta$, it can be assumed that the receiving node has failed.

Nodes may recover from failure. However no node can recover from failure in an undetected manner. If a failed node recovers then at least one alive node will detect the recovery of the node within a fixed time delay after it has recovered. This assumption is the dual of the failure detection assumption discussed above.

For each event occurring in the system a time is associated which is the time at which

the event occurred. All events are recorded in order of their occurrence. In case of any two events occurring at the same time, their order is determined in some predefined manner(for instance, order may be based on increasing order of processor number) known to all nodes.

In case of a node failure, the failing node does not retain any of the previous information. Only the stable storage information remains uncorrupted. The information of the system modeled as the triplet $(V(S), E(S), T(S))$ as defined above is in the stable storage. This information can be used on recovery of a node from failure. On recovery a node should know the initial configuration of the system , the paths to all other nodes in the system and the information of nodes it can test.

2.2 Reliable Broadcast(RB)

RB means that if any message is broadcast by a node i at time T, it should be received by all other nodes that are alive within some fixed time D after the message has been broadcast. RB property is needed by a large number of applications in distributed systems, and even the broadcast media(like the ethernet) do not usually support reliable broadcast. Hence protocols are needed to support reliable broadcast. A number of algorithms have been suggested[4][5][13][16][17], which aim at achieving RB. Some of the different algorithms include Fault Tolerant Broadcast(FTB)[16], Trans protocol for broadcast[17], Total protocol for broadcast[17].

A reliable broadcast protocol ensures that if a node initiates a reliable broadcast, it's message will be received by all other alive nodes in the system. Typically a copy of the reliable broadcast protocol runs at each node with special instructions for the initiator. We assume that broadcast is initiated at time T by a node and it is complete by time $T + D$. For reliable broadcast we classify the nodes into two types:

1. Initiator node : It is the node that initiates the reliable broadcast of a message. The reliable broadcast is specified using the predicate $Sent(m, T, i)$, where i is the initiator node.

2. Receiver nodes : These consist of all alive nodes in the configuration except the sender. The receiver nodes receive the message that has been broadcast by the initiator node.

For reliable broadcast the following property holds:

$$RB \equiv Sent(m,T,i) \Rightarrow \forall j[Received(m,T+D,j) \wedge Alive(j,T,T+D)]$$

where $Sent()$, $Received()$ and $Alive()$ are predicates and are defined as :

- $Sent(m,T,i)$ is true if a node i has broadcast a message m at time T.
- $Received(m,T,j)$ is true if a node j has received a message m by time T.
- $Alive(j,T_1,T_2)$ means that a node j was alive for the duration T_1 to T_2.

The RB property defines the global view regarding the message sent by the initiator of the system, D time units after the RB is initiated. It states that if a message is sent by a node i at time T then all nodes receive the message latest by time $T+D$ provided they are alive from T to $T+D$.

2.3 Membership

The purpose of the membership protocols is to reach an agreement among all members(all alive nodes in the system) of the configuration on the status(failed or alive) of all the nodes in the system. A membership protocol is typically initiated by the node that detects the failure or recovery event of another node in the system. The node that initiates the membership protocol is the initiator. A membership protocol constructs a membership list at all alive processors i.e. it constructs and maintains a set of all alive processors in the configuration at time T, where T is the time when the membership protocol is initiated. Different membership algorithms have been suggested[7][11][12][14], some of which include Membership protocol using Partial Order[11], Membership by Periodic Group Creation Protocol[7], Membership by Attendance List Protocol[7], Membership Algorithms for Asynchronous Distributed Systems[12].

Membership property states that the membership list at all alive processors has the same view of the system's membership, some fixed time units D_2 after the status change of any node is detected. D_2 time units is the amount of time that is needed to register the change in status of the node whose failure or recovery has been detected. This change is registered at all processors that are alive from T to $T+D_2$. Membership Property(MP) is formally stated as below :

$$MP \equiv StatusChange(S_1,S_2,i,T) \Rightarrow$$
$$\forall j[RegisterChange(S_1,S_2,ML_j,T+D_2) \wedge Alive(j,T,T+D_2)]$$

where :

- $StatusChange(S_1, S_2, i, T)$ is the change in status of a processor from S_1 to S_2 as detected by node i at time T.
- S_1 and S_2 are the different states in which a processor can be, where $S_1 and S_2$ are not same.
- $RegisterChange(S_1, S_2, ML_j, T + D_2)$ means that membership list (ML) at node j has registered the change in status from S_1 to S_2 by time $T + D_2$.
- $Alive(j, T, T + D_2)$ is true if node j is alive for the time T to $T + D_2$.

Defining the membership property in this manner requires that the initial condition for membership should be specified. The initial condition for membership should be such that the membership list at all processors should be same when no change in status has been seen for time D_2 i.e. $\forall i, j ML_i = ML_j$ where i and j are alive. We assume that the initial membership list is defined as a set containing all processors in the list and $\forall i, j, ML_i = ML_j = \{0, 1, ..., N - 1\}$.

The membership list keeps changing as the processors keep failing and recovering. In order to clarify the notion of a stable state we assume that a new change is initiated only after a previous change has been recorded at all processors. The system reaches a stable state D_2 time units after a change has been detected. The stable state is the state that is reached when all previous changes have been recorded and no new change in status of any node occurs. As soon as a new change in status of a node is detected the membership protocol is initiated and the system no longer remains in stable state for D_2 time units. When any new change in the state of a node is detected at time T, the node which detects the change will initiate the membership protocol. The membership protocol causes all processors that are alive to reach an agreement on the state of the node in which the change was detected. This change is reflected in ML of all nodes by time $T + D_2$. This is ensured by the membership protocol. As the initial ML is same, the new ML after the change is recorded will also be same for all alive processors, if the membership protocol is functioning correctly.

A number of properties for membership have been described in the following discussion. The basic property of a membership protocol that is achieved by almost all membership protocols has been derived from the basic definition of membership. It is the agreement

property which states that, all status changes in the system regarding failure and recovery of nodes, will be agreed upon by all alive nodes in the system. This means that the information at all nodes regarding membership of the system is the same when the system is in steady state. There are certain additional properties[9] that a membership protocol may be required to achieve. Of the existing membership protocols some protocols can attain certain membership properties. The different membership properties include liveness, accuracy, confidence, partition handling and ordering properties. These properties have been described in the following discussion.

1. Accuracy: It is the property in the membership service that ensures that, any status change(failure or recovery)that has been detected has actually occurred. This means that there is no such case that a status change that has not actually occurred and has been falsely detected. In most of the membership protocols available, the recovery detection is accurate. This is because the recovery in most of the cases is detected only when the recovered processor itself prompts one of the already alive processors of it's recovery. In case of failure detection, the failed processor cannot itself prompt regarding it's failure, and has to be detected by some other live node. In such a case, the node detecting the failure may not always be accurate, as it's accuracy depends on the type of detection method used. Usually failure detection is done using time outs, i.e. if a node does not transmit or respond to a transmission for a certain time period, it is assumed to have failed. It may so happen that, instead of the node being actually faulty, node there may be some other reason for the delay by the node. Hence failure detection cannot be guaranteed to be accurate.

2. Liveness: Liveness property in a system suggests that, all status changes occurring in a system(failures and recoveries) are eventually detected. In most of the cases it is seen that failure detection is live. This is due to the fact that on failure a node stops transmission. Hence the other nodes do not receive any message from it, thus causing a delay and hence the failure is eventually detected. In case of recovery of nodes, the recovered node conveys the information regarding it's recovery itself, but it is possible that due to some other error the information does not reach any alive node. Hence recovery detection cannot be guaranteed to be live.

3. Confidence: Confidence property in a membership service suggests the amount of

certainty in a status change that has been detected. It varies from one protocol to the other. In order to increase the confidence in the decision taken, certain voting algorithms are used before taking the decision, that vote in favour or against the decision taken. Depending on the degree of confidence required, decision can be based on a single node suspecting a change, a majority of nodes voting in favour of the change suspected, or all nodes requiring to vote in favour of a suspected change, in order to confirm it.

4. Partition handling: This is the membership property that decides how system partitions and merging of partitions are to be handled in systems, in case of failures and recoveries. Some systems assume that no network partitions occur. For those systems that allow system partitions, they are handled in different ways. If a node in a system fails, it may cause the system to partition into two subnetworks. Some protocols may cause each of the subgroups formed to independently work correctly as two separate groups. Some other protocol may treat all nodes of one such group as failed, and allow only one group to exist, with all nodes of other subgroup assumed as failed. This could be said equivalent to the case that assumes no partition, as members of only one subgroup are alive. In case of recovery of a node that had caused a partition, the new group may be formed in different ways. The common group may be formed such that all nodes that were assumed to have failed join the group of live nodes one at a time, or all nodes of the two partitions merging to form a common group.

5. Ordering properties: Different membership protocols cause the membership messages to be ordered in several different ways. The types of message orderings done are discussed as follows. Messages can be ordered in FIFO order or can be placed in a total order. Ordering messages in FIFO order means that a stream of messages coming from a node should be received at all other nodes in the same order as transmitted by the node. There is no restriction on ordering of messages relative to messages from other nodes i.e. order of messages from one node is independent of order of messages from other nodes. Total order implies that all membership messages from different nodes should form one single sequence of messages in order of their occurrence, and this order should be same at all nodes.

   Certain applications require that all events as failure, recovery and application mes-

sages be ordered relative to each other. This needs a combine ordering on membership messages and application messages and based on the requirement of the application these ordering properties vary. Certain applications require that on detection of a failure, the last message that is to be broadcast by an application from the failed node should be agreed upon. All nodes that are alive should have the same message as the last message by a node before it failed. Any other message from the failed node sent after the agreed upon last message is not accepted by any node. Another membership ordering properties suggests that the membership change at all nodes occur at the same point in the application message stream. This means that a total order on messages regarding membership change and application messages should exist at each node, and this total order should be same at all nodes. This property has been defined as the virtual synchrony property. Similar other membership properties have been suggested, which include agreement on a set of predecessors before failure, agreement on first message after recovery, extended virtual synchrony property which is an extension of the virtual synchrony property. These properties may or may not be fulfilled by each membership protocol depending on the membership protocol used.

2.4 Diagnosis

The purpose of the diagnosis protocol is to determine the fault condition(status) of all the nodes in the system. A complete diagnosis protocol returns a fault situation FS which defines the fault condition of all nodes at time T. A fault condition can be considered as $(s_0, s_1, s_2, ...s_{N-1})$ where each of s_i is a binary value specifying the condition of node i. In a distributed system the general goal of a diagnosis is such that each node is able to independently determine the state of all the nodes and this information is consistent at all the nodes. This fault situation is reached at by executing the diagnosis protocol on the test results of all the nodes in the system. Several algorithms have been suggested for diagnosis in distributed systems[1][2][3][8]. Two of these include The NEW_SELF Algorithm[1][8], and The EVENT_SELF Algorithm[2].

In these algorithms each node should be able to test some other nodes to achieve diagnosis. The information regarding which node can be tested by a certain node is typically specified by the model in the diagnosis protocol. As we are assuming fail-stop processors, faulty processors will not test any nodes. All test results that are obtained are those from

alive nodes and hence will be correct.

The diagnosis protocol runs at each node that is alive. Initially all nodes are assumed to be alive so as to maintain consistency at the time of starting the system. The alive nodes then can determine the actual situation after the first diagnosis. Diagnosis protocol is initiated by the node that wants to determine the fault situation of the system S. If T is the time when the diagnosis protocol is initiated then by diagnosis property :

$$DP \equiv FS_g(T) \Rightarrow \forall i,j[equal(FS_i, FS_j, T + D_3) \wedge alive(i, T, T + D_3) \wedge alive(j, T, T + D_3)]$$

where:

- T is the time when system diagnosis is started.
- D_3 is the time taken by system diagnosis to complete.
- $FS_g(T)$ is the global fault situation of the system at time T.
- $alive(i, T, T + D_3)$ means that the node i is alive from time T to D_3.
- $equal(FS_i, FS_j, T + D_3)$ is true if by time $T + D_3$ the fault situation FS as determined at node i and at node j is the same.

At the completion of system diagnosis all alive nodes i have a set of values for all nodes in the system $FS_i(T + D_3)$ that tell the fault situation of the system at time T. In FS_i if the value of $s_i = 0$ it means that the node i is fault-free. If $s_i = 1$ the node i is faulty.

The diagnosis property states that the fault situation of the system S as tested at time T will be reflected at all alive nodes by time $T + D_3$. For achieving diagnosis each node performs tests on certain other nodes and communicates the test results such that all nodes have results of all tests performed by different nodes. The tests are performed by alive nodes only and hence the test results give correct information. Test information gathered at all nodes is the same, hence the system diagnosis achieved from these results will also be same at all nodes. Hence if the diagnosis protocol is correct the diagnosis property will be fulfilled.

Chapter 3

Protocols and their Analysis

In this chapter, a number of existing protocols for the properties discussed have been described. The complexity for each of the protocols has been determined in terms of number of messages used by each, and the time taken. A comparison table that contains the complexities of various protocols is given.

3.1 Reliable Broadcast Protocols

3.1.1 Fault Tolerant Broadcast(FTB)

It is a distributed program that ensures that all alive nodes will receive the message that has been broadcast by any one of the alive nodes. This protocol assumes reliable communication among nodes and fail stop processors. Communication among processors is through the exchange of messages and acknowledgments. For the fault tolerant broadcast protocol discussed, the system is seen as an ordered tree with a root node r, the nodes and edges of the tree depicting processors, and direct communication links between processors respectively. All broadcasts are initiated at the root node r. In order to ensure a reliable broadcast, it has to be established in case any node fails. We establish the reliable broadcast for two cases i.e. when any node except the root node fails, and the second case is when the root node may also fail.

In the first case when the root node does not fail, any other node or nodes can fail. To achieve reliable broadcast in such a situation, the root node r sends a message to all it's child nodes and waits for the acknowledgment from all of these. The children on receiving the message, send it to their children and so on. When a node receives an acknowledgment from all it's children, it sends back an acknowledgment to it's parent. If a node fails the message may not reach it's successors. In such a condition the parent of the failed node

sends the message to all successors of the failed node. When the root node i.e. the initiator node receives an acknowledgment from all it's successors it means that the reliable broadcast is complete.

For the case when the root node r fails, if it has failed before sending the message to any other node, then this means that the broadcast has not started and hence there is no inconsistency of information at different nodes. If node r sends the message to a node i and then fails, the node that receives the message must establish the reliable broadcast. In case more than one node has received the message, then all these would establish reliable broadcast. This does not cause any problem except that some nodes may have to receive the message, and acknowledge it more than once.

Different variables are used by the protocol in order to keep track of the messages to be sent, acknowledgments to be sent, and acknowledgments to be received by all nodes. The various variables used include send_to, ack_to and ack_from.

- send_to : It is a set of processors at each node, from among those, to which the node must send the message and to which message has not yet been sent.
- ack_to : It is a set of processors at each node which have sent the message and to which ack has yet to be sent.
- ack_from : It is a set of processors at each node from which ack is awaited.

The variable send_to keeps a list of all processors to which it has to send the message. For a node i it will initially contain all $succ(i)$ in it. If one of the $succ(i)$ fails then all it's children are also included in the set send_to. Once the message is sent to a node, it is removed from the set send_to. This means that a node sends the message at the most once to every node in the process of one broadcast. Similarly for set ack_to, as soon as an ack is sent for a message received. The node to which ack has been sent is deleted from the set ack_to. This means that any node receives at the most one ack from a given node for one broadcast. It can be seen that in the FTB protocol just described, the total number of messages needed to achieve a reliable broadcast include the messages and the acks sent. Each node sends at the most one message to every other node and receives at the most one acknowledgment from every other node in the network. This concludes that for a system with n nodes the maximum number of total messages (including messages and acknowledgments) to achieve a RB are $2n(n-1)$. The number of messages needed in case when no failures occur will be

much lesser. For such a case all nodes will transmit a message to their children nodes, and receive an ack from them. For a system with an average of c children per node, the total number of messages needed will be less than $2n * c$.

Time taken to achieve a reliable broadcast depends on a number of parameters. These include the time taken to transmit a message from one node to another, the average number of children that a node has in the system's tree that is used in the protocol, and the total number of nodes in the system. If T_p is the time taken by a node to communicate a message from one node to another, and c is the average number of children that a node has, then for a n node system the time taken for a message to reach all nodes can be calculated. Each node will transmit the message to it's c children, and if we assume the processing time as negligible, all the children of a node will receive the message simultaneously. For n nodes there will be $log_c n + 1$ levels in the tree. The time taken for the message to reach all the nodes will be approximately $T_p * (log_c n + 1)$. Amount of time required to receive the acknowledgments will be the same. Thus we can say that the total amount of time needed for a reliable broadcast for the FTB protocol is $2 * T_p * (log_c n + 1)$.

3.1.2 Trans Protocol

The system model consists of a reliable communication with fail stop processors. The trans protocol is a reliable broadcast protocol where the messages are not acknowledged immediately after they are received. The acknowledgment is piggybacked on a new message that has to be sent by any of the nodes that have received the message. All other nodes that have received the first message, on seeing it's acknowledgment on the second message, no longer need to send an ack for the first message. In case when a node has not received the first message, it sees the ack on the second message and hence determines that it has not received the message. In such a case it sends a piggybacked negative acknowledgment for the message it has not received, when it has a message to broadcast. Hence by the trans protocol it is possible to achieve RB, as any processor that is alive and has not received a message can request for a retransmission by sending a nack as soon as it detects that it has not received a message. If a node accumulates a number of messages for which it has to send acks it sends a dummy message in order to acknowledge the messages received.

If a processor does not receive an acknowledgment, a fixed time delay after sending the

message, it assumes that the transmission was unsuccessful, and hence adds the message to it's retransmission list. A processor keeps the message until it knows that all other processors have received the message.

In the trans protocol we have seen that to achieve a reliable broadcast a minimum of one broadcast of a message is required. No special message is needed by the protocol to confirm whether a message has been received or not, as all acks and nacks are piggybacked. For a n processor system, if we assume that for a broadcast, all processors receiving a message send a copy to all other processors, it would mean that a maximum of $n * (n-1)$ messages will be needed to complete a broadcast. As acknowledgments are piggybacked no separate messages are needed for them. Hence in comparison to the fault tolerant reliable broadcast protocol, half as many messages are needed in the above protocol. In case of a retransmission the number of messages is increased by the number of retransmissions done.

Time taken by the protocol depends on the time taken to place the messages in the total order, before committing them to the application. In this algorithm, for a message to be placed in the total order, it is required that, the message should be followed by $|(n+2)/2|$ number of messages, all from different processors. This is the case for single resilient systems. In case of higher resiliency, say x, the number of messages needed are increased to $|(n+1+x)/2|$. We assume an ideal system that always transmits a message without requiring a retransmission. Some node is always ready to transmit a message. Also the system is such that all nodes wish to send a message with equal probability, so that if n messages are transmitted then the probability that they are all from different processors is one. If we make these assumptions, then it will take $T_b * |(n+2)/2|$ units of time, for a message to be followed by $(n+2)/2$ messages. The above case is the ideal case, the probability of which in a real system would be low. Depending on the system's behaviour regarding the transmission of messages, the time taken to place a message in total order varies. The average time will be much lesser as, for every new message from a different processor it will be possible to place a message in the total order.

3.2 Membership Protocols

3.2.1 A Membership Protocol using Partial Order

This protocol uses the psync multicast mechanism as the basis for communicating among processors. Psync provides a medium for conversation that allows processors to maintain

a partial order on the messages that are exchanged. Initially all the processors of the system are included in the membership list(ML). Later on, due to failure of processors, and due to recovery of failed processors, the membership lists at all processors are modified. New messages are sent in context of messages already sent or received. The order of these messages at each node is specified by a graph called the context graph, where the vertices of the graph represent the sender of the message with the sequence no. of the message. An edge from one vertex to another specifies that the second message has been sent in context of the first message. Any message received is attached to the graph depending on it's context. All messages sent at the same logical time constitute a wave. A message is said to be stable if it is followed by a message from all the nodes in the system. If the previous message is not received by a node, it will be discovered when a new message in it's context is received.

For the purpose of determining membership, the protocol is initiated when a failure or recovery occurs. These two events are detected by a separate protocol called the detection protocol. On initiation of the membership protocol, the message regarding failure or recovery is sent to the conversation. It will be seen by all processors that are alive. Hence all processors get the information of a failure or recovery, and hence the ML is updated accordingly. ML at all processors will be same some time after every failure or recovery, if no new event occurs.

The protocol maintains two lists that keep information regarding processors that are suspected failed or recovered from a failure. *SuspectDownList* is a list of all ***p is down*** messages that have been received, and *SuspectUpList* is a list of all ***p is up*** messages. The lists become empty when the suspected failure or recovery are confirmed. Different logical times have been defined regarding the failures and recoveries of different processors. Suspected down time is the logical time i.e. the time corresponding to the wave containing the ***p is down*** message. Actual down time for a failed processor p is the time corresponding to the wave at or after which no message from p has been received. Realized down time is the time corresponding to the wave when the failed processor is removed from the membership list. Suspected up time is the time corresponding to the wave containing the ***p is up*** message. Realized up time is the time corresponding to the wave when the recovered processor is included in the membership list. Membership check state is defined as the state when any of the suspect lists are non empty i.e. either a failure, or a recovery, or both failure and

recovery are being suspected. Membership check period is the time period for which the system is in the membership check state.

In case of single failure or recovery, the protocol works as follows. If the detection protocol detects the failure of a processor p, it issues a *p is down* message. Any such message received is included in the *SuspectDownList*. When the detection protocol suspects a recovery it issues a *p is up* message which is added to the *SuspectUpList*. If any message that negates the *p is down* message i.e. *nack p is down* is received immediately after the *p is down* message, then it is removed from the *SuspectDownList*. A nack is sent by a processor only if it has received a message from the suspected failed processor at the same logical time at which the *p is down* message has been issued. If the messages in the *SuspectUpList* become stable then the corresponding processors are added to the membership list. When all members in *SuspectDownList* become stable the corresponding processors are removed from the membership list. When there are no messages in either of *SuspectDownList* and *SuspectUpList*, then the membership check state ends. This means that the system is in stable state, and will remain to exist in stable state unless some message is added to these lists.

In case of multiple failures or recoveries, if a failure occurs a suspect message is issued by the detection protocol. If a second suspect message is received before all nodes send back an ack for the first, in such a case these nodes cannot participate in each other's decision. Both nodes remain in the suspect list until acks from all remaining alive nodes are received for both. The system remains in the membership check state as long as any suspect message remains in the suspect list. The properties of the protocol have been discussed as follows.

- Accuracy and Liveness : The recovery and failure detection in this case will not be accurate, as the detection is done by the detection protocol. The failure detection is live i.e. any failed node will eventually be detected as failed.

- Agreement : All alive nodes in the system agree on any status change, and the agreement occurs at the same logical time all the nodes. The membership lists at all alive nodes is same in stable state i.e. when the system is not in the membership check state.

- Confidence : In this case the degree of confidence in the status change is high. This is because even if a single node does not agree with the suspect message, it sends a

nack, and the status change does not take place.

- Partition handling : The above protocol assumes that the network always remains connected i.e. all alive nodes can interact with each other despite any number of failures. Hence this protocol does not specify any partition handling property.

- Ordering : In this protocol, a context graph of messages sent in context with other messages is constructed. This graph decides a partial order on messages from all different processors as soon as the messages are received. But eventually when all messages are received a total order can be constructed which is same at all nodes.

In order to find the complexity of the algorithm, we determine the number of messages needed to achieve membership and time taken for the same after a failure or recovery has been suspected. Certain assumptions have been made in order to simplify the process. We assume that a message takes a maximum of T_m time units in order to reach a node after it has been sent. Message processing time is assumed to be negligible. Reply to a message is sent without delay, immediately after receiving a message. We are assuming the case of single status change at a time i.e. to determine the complexity we deal with single failure or recovery.

As described in the protocol, the detection protocol issues a *p is down* or a *p is up* message when it suspects a failure or a recovery. If the maximum number of nodes in the network is n, and we assume that all nodes are alive then for a suspected failure, the remaining $(n-1)$ nodes respond with a nack or an ack message. If all ack messages are received the node is removed from the membership list. The total of n messages used to remove a failed node from the membership list. In case of a recovery message, if all but one processor is alive then all $(n-1)$ processors send an ack, before the recovered node can be added to the membership list. The maximum number of n messages are used to include a recovered node to the membership list.

In order to determine the time taken for a status change to be registered, time taken after the detection protocol has suspected a change is to be calculated. As we are considering only single failure or recovery, the membership check time will be the time taken for a single status change. We assume that on an average, a wave takes T_m time units to complete. The suspect message is sent to the conversation as soon as it is detected. This message will belong to the first wave which will take at most T_m time units. The ack message, or

the nack message in case of failure, is sent by the nodes, and this is sent in context to the suspect message. All these messages belong to the same wave, as they are sent in context to the same message. These are sent immediately after receiving the suspect message. This wave also takes at the most T_m time units in order to be complete. Only on receiving an ack from all nodes, the membership list changes. Hence if a status change is detected at a time T, then The membership will be achieved at $T + 2T_m$.

3.2.2 Periodic Group Creation Protocol

The system model consists of the processors connected by the physical communication medium, with a membership server on each processor. The network supports reliable point to point communication. A maximum delay of d time units is assumed for transmission of a message m from one node p to another node q, where (p, q) is a direct link in the network. To achieve the broadcast, a message diffusion service is assumed i.e. the message to be broadcast is transmitted in parallel on all available links and is forwarded from one node to the next. The message diffused by a node is received by all other nodes within a fixed time delay N, known as the network delay which depends upon the point to point delay, maximum network diameter and message processing delays.

If n is the total number of nodes in the network and d is the average degree(number of neighboring nodes of a node) then a broadcast of a message by a node would require approximately $n * d - n + 1$ point to point messages.

In the periodic group creation protocol, when a new processor starts, it initiates it's membership server in order to form a group. Even for joining an existing group, a new group has to be formed so as to include the new processor. For the creation of the new group, the membership server of the processor broadcasts a 'new-group' message to all processors. If the other processors wish to form a group with a new processor, then on seeing the 'new- group' message they respond by broadcasting a 'present' message. A 'present' message contains the identification of the processor that has sent it. When a 'present' message is received from all different processors a fixed time after it had been sent, a new group is formed. Since all processors receive the 'present' messages from all other processors, a membership list is constructed by all processors independently. This membership list corresponds to the time, when all processors had received the 'new-group' message.

For a processor wishing to join a group, it can itself initiate the new group procedure as described above. In case of a failure, the failed processor is unable to communicate because the system model assumes failed- stop processors. Hence the failed processors are not able to help in determining the new membership after a failure. In such a case the processors that remain alive have to determine the new membership. In the periodic group creation protocol the failures are handled by using the concept of group renewal time. This implies that every group that has been formed remains a valid group only for a certain period of time after it's creation. As this period ends the group is no longer a valid group unless it's membership is renewed. So if any processor fails at some point of time, it's failure is detected at the group renewal time and it is excluded from the new group. To implement the group renewal time, a time period is fixed which is known to all nodes and after which every member is supposed to broadcast a 'present' message in order to prove it's presence. As all 'present' messages include the processor id's, their presence can be determined. Any processor failing to send the present message is assumed to have failed. A new group excluding the failed processor is created after detecting the failure by the group creation process. The properties for the protocol have been discussed as follows.

- Accuracy and Liveness : The recovery detection in this case is accurate as the recovered processor itself sends a message for it's recovery but it may not be live. This will be the case when the recovery message does not reach any of the already alive nodes. Failure detection will not be accurate, as it is detected by a time out of the renewal time period. in case the processor is slow and is unable to send a renewal message in time it is detected as failed though it is alive. The failure detection is live i.e. any failed node will eventually be detected as failed.

- Agreement : All alive nodes in the system agree on any status change, and the agreement occurs at the same time at all nodes, because as soon as a status change is detected, a new group is constructed, all nodes of which agree on the group membership.

- Confidence : Each node confirms it's own recovery, and failure detection is by time out. No voting is done in order to reach a decision. Hence the degree of confidence is low.

- Partition handling : The protocol allows multiple group formations, in case no commu-

nication between two groups of nodes is feasible. All nodes interact with the members of it's group without any problem and all groups have the same status. On recovery of a node that wishes to communicate to members of more than one group, a new group is formed joining the members of both the groups.

- Ordering properties: The decisions on failure are taken at the group renewal time, hence all such failures are detected at the same point in time. The underlying system model is synchronized, and messages can be timestamped at each node and a FIFO order of messages maintained.

To evaluate the performance of the protocol suggested above, the number of messages used by the protocol for join handling and for failure handling for a n processor system have been evaluated. At the time of forming a group, one broadcast is needed for the 'new-group' message. In response to this message, all n processors broadcast a 'present' message. Hence a total of $n+1$ broadcasts are needed for creating a new group or for adding a new member to a previously existing group. As given in the system model, one broadcast needs $(n*d-n+1)$ point to point messages, where d is the average degree of a node. This makes a total of $(n+1)(n*d-n+1)$ point to point messages needed for membership in case of a group creation or a join handling.

The number of messages needed for failure handling are the same as those needed for renewal of a group. For membership renewal, all n processors broadcast a 'present' message. If a node has failed, it's failure will be detected. If a failure is detected a new group is created excluding the failed processor from the group. Membership renewal needs $n(n*d-n+1)$ point to point messages. In case of failure of a node a maximum of $n(n*d-n+1)$ point to point messages will be needed.

Evaluation of the time taken by the protocol in order to create a new group will depend on the time that is needed to complete a broadcast. Assuming that a broadcast takes T_b units of time, we can evaluate the time needed to achieve membership. Processing time for each message is assumed to be negligible. For a group creation one broadcast is needed for the 'new- group' message taking time T_b, and on seeing it all nodes simultaneously broadcast a 'present' message that also takes the same amount of time i.e. T_b. This means that a total of $2*T_b$ time is needed for new group creation and for join handling. If a node fails, it's failure is detected only at the group renewal time. A node may fail immediately after

sending a 'present' message for a previous group renewal. It's failure will be detected at the next group renewal time. If the group renewal time is T_r then a failure takes at the most $T_r + T_b$ time units when processing time is negligible. This includes the group renewal time T_r and the time needed to broadcast the 'present' message T_b.

3.2.3 Attendance List Protocol

This protocol is a modification of the periodic group creation protocol. In this protocol an attempt to reduce the number of messages required has been made. This protocol creates a new group, only when a failure or a recovery of a node takes place, unlike the periodic group creation protocol where a new group is created every time a the membership is renewed. Join handling is the same as in the periodic group creation protocol i.e. every time a new processor wishes to join, it broadcasts a 'new-group' message, to which all processors wishing to join respond, and a new group is formed.

For handling the failures the protocol has been modified. In order to check the actual membership of the group some fixed time after the group has been created, a membership check time is decided upon. After every membership check time, the group membership is checked by issuing a list of all members of the group, which each member is required to sign, in order to validate it's presence. This list is issued by one processor which is decided upon by some selection procedure known to all processors. All processors on receiving the list, sign it and relay it. All processors should receive the list within some fixed time T_c, after the membership check list is sent. This is known as the membership confirmation time. If a node does not receive the list of members for signature before the next membership confirmation time it assumes that some node has failed and it issues a 'new-group' message in order to form a new group excluding the failed processor. When the list reaches the initiator, if the signatures of all members are present, then the membership is the same, otherwise a new group has to be formed. The new group creation is done similarly as the joins are handled. The advantage of this protocol over the group creation protocol is that this protocol needs just n messages in every membership check time period if no failures occur in this period. The membership properties are the same for this protocol as for the periodic group creation protocol.

The number of messages used by the protocol for join handling is the same as that in the periodic group creation protocol. For failure handling the number of messages needed

vary for the two cases i.e. for checking failures in steady state and for creating new group in case of a failure. As seen in the protocol, in order to check the membership n point to point messages are needed in order to relay the membership check list. In case of failures, the node detecting the failure initiates a new group creation, which would need $(n+1)$ broadcast messages. A total of n messages and $(n+1)$ broadcasts will be needed in case an actual failure occurring. This makes a total of $(n+1)(n*d-n+1)+n$ point to point messages needed for membership in case of a failure handling.

For a join handling the time taken is the same as in the periodic group creation protocol. This means that a total of $2*T_b$ time is needed for new group creation and for join handling. If a node fails, it's failure is detected at the membership check time. A node may fail immediately after sending a 'present' message for a previous group check. It's failure will be detected at the next group check time. If the group renewal time is T_r then a failure is detected at the most $T_r+2*T_b+T_c$ time units after it actually occurs. Message processing time is assumed to be negligible. This includes the group renewal time T_r, the membership conformation time T_c and the time needed to create a new group i.e. $2*T_b$, after the failure is detected.

3.2.4 Membership Algorithms for Asynchronous Systems

Two algorithms for asynchronous distributed systems have been described. These are membership algorithm A and membership algorithm B. The system model consists of a set of distributed processors, each processor is referred by their identifiers, and they communicate by broadcasting messages. A processor that wishes to join an existing group is termed as the applicant. The time for which any processor is in a group, i.e. from it's time of joining to it's time of removal, is an incarnation. An existence of a total order on all messages is assumed and this is same at all processors. Each message has an associated ordinal number depending on it's position in the total order. Different functions that are defined with reference to the total order are said to be functions of the ordinals. These functions are evaluated by the processors based on the ordinal of the last message placed in the total order. In the following algorithms, membership $m(x)$, cardinality of membership set $n(x)$, and resilience $k(x)$ are all functions of the ordinals. $m(x)$ is the membership of the system, as determined after message with ordinal x is placed in the total order, and before the next message is placed in the total order. $n(x)$ gives the cardinality of the membership set and

$k(x)$ is the resilience of the system. The system configuration at ordinal x is defined by the triplet $m(x), n(x), k(x)$.

Membership Algorithm A -Three types of broadcast messages are needed by this algorithm. A request message which is a special message broadcast by the applicant processor. It contains the identifier of the applicant processor. A sponsor message which is an ordinary message and can be broadcast by any number of processors in the network. It includes the processor identifier of the applicant and also the incarnation number that has been proposed for it. Grant message is a special message broadcast by all processors in the system. It includes the ordinal x of the sponsor message in the total order and the $m(x)$, $n(x)$, and $k(x)$ values.

The processor wishing to join the configuration broadcasts a request message, and continues broadcasting it unless it is included in the configuration. A processor on receiving a request message determines if it is a new request message. If the request message is new, then the incarnation number for the requesting processor is determined, and this is broadcast in the sponsor message by all processors receiving the request message, if they have not yet seen a sponsor message. The sponsor message also contains the applicant's identification. A total order on all messages is constructed. The ordinal say o of the first such sponsor message is extracted from the total order and is placed in the grant message. This grant message is sent to the applicant and it also contains all information for the applicant processor so that it can update it's information on joining. The applicant on receiving majority of grant messages with incarnation as determined by the processor issuing the sponsor message joins the configuration at ordinal o. If there is any message that had been ordered after the sponsor message and broadcast before the grant message, then it would not be received by the applicant processor. In such a case a processor can ask for a retransmission after joining.

In this algorithm the removal of a failed processor does not require grant messages. A processor is said to have failed if it either fails to receive a message or if it fails to broadcast a message at ordinal x. Once it is determined that a processor has failed, the processor that detects the failure requests it's removal. In this algorithm the request message does not include enough information to suggest that it is a request for a failure. If a processor is a member of a configuration and none of it's messages have been ordered, then it is not possible to decide whether the request is a continuation of a previous join request or a

request for a failure. In such a case, to detect the failure, the processors will have to wait until it has been found that the processor in question has not broadcast a message, before it can be removed. The failed processor is removed at ordinal x, i.e. when the request message for removal is placed in the total order.

The complexity of this algorithm can be calculated in terms of number of messages required in order to achieve membership. For algorithm A the number of messages needed include one request message, one sponsor message and n grant messages. As given in the protocol, request and grant messages are special messages and sponsor message is an ordinary message. This means that a total of $1_o + (n + 1)_s$ messages are needed to be broadcast in order to achieve membership.

In order to determine the time taken to achieve membership, it is required to determine the time taken to place a message in the total order. We assume that the average latency in order to place a message in total order is T_t. For algorithm A it is required to construct a total order on the sponsor message. Each node is also required to receive a majority of grant messages which would need a little less than T_t time. Hence a maximum of $2 * T_t$ time units(approx.) is the time needed by the membership algorithm A to achieve membership.

Membership Algorithm B -This algorithm assumes that the prompt delivery of messages is recognizable. In this algorithm the sponsor message is replaced by reference message. Here a majority of grant messages are also not required. Any processor that wants to join the configuration, instead of broadcasting a request message waits for an ordinary message that has been sent by some member, and that the new processor can recognize it as having been promptly delivered. On receiving such a message the applicant includes the identifier of this message in it's request message and broadcasts it. The reference message acts as the sponsor message as it's identifier along with the identifier of the applicant is included in the request message. On receiving the request message, which includes the identifier of the reference message, the receiver waits till the reference message is ordered at an ordinal x. The receiver then determines an incarnation number for the applicant processor. Then it sends a grant message to the applicant which includes the incarnation number of the applicant, and all other information needed by the applicant to build a total order. On receiving the grant message the applicant orders the first such message at ordinal z and is included in the group at this ordinal.

For failure handling in membership algorithm B, as soon as the failure of a node is detected by a processor, it broadcasts a request message. Unlike algorithm A, the request message in this algorithm is enough to ensure that it is a request for a failure. Hence the failed processor is removed from the configuration from ordinal x, i.e. when the request message for removal is placed in the total order.

The number of messages required in order to achieve membership using algorithm B have been calculated as follows. For algorithm B, the number of additional messages needed include one request message and one grant message. No sponsor messages are needed in algorithm A, as the already existing ordinary message that has been broadcast by some node, is used instead as the reference message. As given in the protocol, request and grant messages are special messages, which means that a total of 2_s additional messages are needed to be broadcast in order to achieve membership.

As in the algorithm A, it is required to determine the time taken to place a message in the total order. We assume that the average latency, in order to place a message in total order is T_t. For algorithm B after receiving the request message, it is required to place the reference messages in the total order, so as to determine it's ordinal. This would need T_t time units. After determining the incarnation number for the applicant, the grant message is broadcast by the nodes. It is required to place all grant messages received, in the total order so as to determine the ordinal of the first such message which would need another T_t time units. Hence a maximum of $2 * T_t$ time units(approx.) is the time needed by the membership algorithm B to achieve membership.

The membership properties for the asynchronous protocols are as follows.

- Accuracy and liveness: The assumptions made by the protocol state that all membership change messages will eventually be received by some alive processor, placed in the total order, and required action taken with a probability greater than zero. Hence we can say that the membership protocol is live. The recovery messages are sent by the recovered node itself hence it is accurate, whereas the failure is detected by some other alive node hence is not accurate.

- Confidence: There is no voting done before confirming the failure or the recovery decision. This means that the degree of confidence is low.

- Agreement: All processors agree to the membership change, some fixed time after it

is detected. This means that the agreement property holds true.

- Ordering: The protocol assumes that an underlying total ordering protocol is used. This allows a total order to be placed on all messages.

- Partition handling: The protocol requires a majority of nodes to be alive in a group for a membership change decision to be taken. The system allows system partitions as long as one group has enough number of processors so as to enable membership. All partitions with lesser no of nodes are assumed as failed. Clearly, there will be no more than one live group after partitioning.

3.3 Diagnosis Protocols

3.3.1 The NEW_SELF Algorithm

The system is modeled as a set of nodes and communication paths between the nodes. Every node possesses the capability to test certain other neighboring nodes, those that are connected by a direct edge, in order to check it for correctness. Test results obtained from fault-free nodes are correct, and can be used for diagnosis. In this system model, failed-stop processors are not assumed. This means that faulty nodes can also perform tests but, the test results obtained from faulty nodes are unreliable. In this model, failures also occur in the transmission facilities provided, i.e. in the communication links. Any fault in a communication path will be diagnosed by the node making use of the communication path. It means that if a node is unable to communicate with another node then the error may be in the node or in the communication link.

The distributed system can be represented by an undirected graph of the system S, with $V(S)$ representing the nodes of the system that correspond to the processors, and $E(S) = \{(P_i - P_j)\}$ the edges, where P_i has the direct communication link to P_j. The testing graph $T(S)$ is a directed graph of the system S with $V(TS) = V(S)$ representing the nodes of the testing graph, and edges $E(TS) = \{(P_i \rightarrow P_j)\}$ which means that P_i can test P_j, and $E(TS)$ is a subset of $E(S)$. Link failure domain(lfd) of a distributed computing network is any single source of failure among the internode communication links available. Link failure domain set(lfds) for a distributed system is denoted by D_s and is the set of all lfd's for the network. Distributed computing network is represented by the triplet (S, T_s, D_s).

The diagnosis algorithm given below determines faulty nodes and links, and also allows previously determined faulty resources to return to the system. In the given algorithm nodes test neighboring nodes for failures or recoveries, and these test results are transmitted throughout the network. The main problem to deal with in this algorithm is that the test results obtained from faulty nodes are not reliable. This problem is overcome by restricting these test results. Only the test results from fault- free nodes are transmitted throughout the network. A node P_i will accept a diagnostic message from a node P_j only if it has tested P_j and ascertained it to be fault-free. The algorithm works as follows :

- Node P_i tests a node P_j for correctness.
- If P_j is tested fault-free by P_i, it sends it's test result and the information it contains to P_i. P_i stores this information in a temporary buffer.
- Node P_i tests node P_j again for correctness. If P_j is still correct, then the information stored in $P_i's$ buffer is validated and used for further transmission to the testers of P_i.

In this way only the valid diagnostic information flows backward along the paths in the testing network.

The diagnosis process encounters difficulty when recovered, or when facilities are to be reentered in the system. This is because it would be required to keep track of the time order of different failures and recoveries. If a node has to be reentered after a failure, it must have a message regarding it's last failure at it's tester node. The order of failures and recoveries should be such that there are no ambiguities. To avoid any such problems each message has an associated time stamp attached to it. Another problem that still remains is that on failure, the failed node loses it's old time stamp.

Two arrays are maintained at each node containing messages for diagnostic purposes. One of these contains the message by a node, regarding the failed nodes and the time of issuing the message. The second array contains the messages by a node regarding alive nodes and the time of issuing the message. This message also tells if a previously failed node whose entry is in the first array has recovered, and the time of issuing the message. The time of failure of a node as given in one array is compared to the it's time of recovery from the second array to check the order of the two messages and determine if the node is failed or alive. In order to limit the size of arrays, the messages are added only if they specify any new information.

The above algorithm also allows for determining link faults. If a node i is unable to test a neighboring node j but gets information from another node k that node j is fault-free it determines that link $i-j$ is faulty. This information is placed in an array for faulty links. When node i is able to test node j the link is known to have recovered and is included as fault- free by removing it's entry from the faulty links array.

In order to determine the complexity of the NEW_SELF algorithm certain variables have been defined. n is the maximum number of nodes in the network, t is the maximum number of nodes that a node tests, f is the number of failed nodes of the maximum allowable failed nodes, and T_c is the time between the test cycle periods. In order to determine the number of messages required to achieve diagnosis, the number of messages to initiate tests, transmit test information and obtain test results is calculated. In this algorithm each node is required to test some other nodes. We have assumed that each node tests a maximum of t nodes. In one test period each node tests t other nodes as a result of which t new test results are obtained every T_c time units. All nodes in the system receive and forward the test result for every test performed. The total messages that a node sends include the t messages to test t other nodes, t messages to send test results to the tester nodes. We are assuming that all information of all tests is carried in a single message which is included in the message containing the result of the test performed. In one test period the tests are performed twice. Hence a total of $n(2t+2t)$ messages are required in order to obtain a diagnosis.

Time taken to achieve a diagnosis depends upon the test cycle period T_c and on the diagnostic diameter of the network which means the largest distance that the diagnostic information should be transmitted in the given network in the worst case i.e. despite a number of failures. For the case where a node tests t other nodes, the diagnostic diameter will be $(n/t+1)$. The total time taken will be equal to the time taken by test result of one node to all other nodes. T_c is the time needed for forwarding the testing information to the neighboring testers. For the information to reach all nodes time taken will be $(n/t+1)*T_c$ which is also the time taken to achieve diagnosis.

3.3.2 The EVENT_SELF Algorithm

It is an adaptive distributed system diagnosis algorithm. The system model consists of a set of processors and communication links represented as an undirected graph, the vertices of

the graph representing the processors and the edges representing the communication links. It is assumed that each node can test certain other nodes. These tests are represented by a directed graph with vertices representing the processors, and the directed edges $(p \rightarrow q)$ for all (p, q) represent that node p can test node q. The tests to be performed are not predefined, but are based on the fault situation at the time of testing. In this protocol it is assumed that the results obtained from the fault-free nodes are correct, but the test results obtained from faulty nodes are unreliable. hence for correct diagnosis it is essential that each node is tested by at least one fault-free node. In the adaptive algorithm described, each node is tested by only one fault-free node. Nodes are tested after every fixed time intervals and diagnostic information based on these test results and that sent by other nodes, is gathered at each node. The steps followed for testing are given below.

1. The tester node i, tests another node j to find it fault-free. If the node tested is faulty, the tester tests another node until it finds a fault-free node.

2. The diagnostic information collected by node j is sent by it to it's tester node.

3. The node i i.e. the tester again tests node j for it's status.

4. If tested node is still alive, the information it had sent is considered to be valid and can be used to achieve correct diagnosis and to forward it to it's tester nodes.

In this algorithm it is assumed that no node can fail and recover from the failure undetected. The time interval between two tests is small enough to detect all failures and recoveries. If the time interval between two tests is large, then some other method can be used to detect if a node fails and recovers between two test periods.

For the purpose of achieving the diagnosis, each node tests it's next higher node in order to determine if it is faulty or fault-free. If a node tests another node as faulty, then it tests the next higher node until it finds a fault free node. The last node tests the first node, thus creating a cycle of tester nodes. The test results and the diagnostic information flow backward, i.e. in the reverse direction of the tests being performed. All nodes receive the test results of all other nodes as all nodes get connected by a cycle. On receiving the information, all nodes perform the system diagnosis independently.

For the adaptive DSD algorithm discussed, each node tests one fault-free node. All test results obtained for the tests from different nodes are forwarded to the tester node in a single

message. We calculate the number of messages needed for a n node system. Each node initiates a test and sends back a test result including the test information obtained from other nodes to it's tester node. The test is performed twice in each cycle for all fault-free nodes. For all faulty nodes an additional test is needed in order to test another node that is fault-free and test results are to be sent. A total of $n(2+2+2f)$ messages is needed where f is the number of faulty nodes.

In order to determine the time taken to achieve the diagnosis for the protocol described, the longest diagnostic diameter i.e. the longest distance that the test information has to flow in order to reach a node is required. In the algorithm a node tests only one other node in case of fault-free nodes. So the longest diameter for a n node system is n. A node in order to receive information of tests performed at all nodes has to wait for n test cycle periods. If time taken by a test cycle is T_c then time taken to achieve diagnosis will at the most be $n * T_c$.

3.4 Comparison

Different protocols for RB, membership and diagnosis properties have been discussed, and their complexities in terms of number of messages used, and time taken to achieve each property, have been determined. These results are compiled in a table and a comparison of different protocols follows.

Table for comparison of the complexities for different protocols.

Protocol used	No. of messages	Time taken
Fault Tolerant Broadcast	$2 * n(n-1)$, (max), and $2n * c$, (min) n is the total no. of nodes.	$2 * T_p(log_c n + 1)$ T_p is the time taken for point to point transmission of a message. c is the average no. of children of a node.
Trans protocol	1 B'cast i.e. $n*(n-1)+r$ point to point msgs. r is the number of retransmissions done.	$T_p(log_c(n+1) * \lvert(n+x+2)/2\rvert$ x is the resiliency of the system.
Membership by partial order.	n point to point messages.	$2*T_m$ is the time taken in case of single failure or recovery at a time. T_m is the time taken for a wave to get stable.
Membership by periodic group creation protocol.	$(n+1)$ b'cast i.e. $(n+1)*(n*d-n+1)$ msgs.	T_r+T_b , T_r is the group renewal time, T_b is the time taken for a b'cast. i.e. $T_p * (log_d n + 1) + T_r$
Membership by attendance list protocol.	$(n+1)*(n*d-n+1)$ point to point msgs. required for create and join and failure handling. n point to point messages to renew membership, in case there is no failure.	$2 * T_b$, for join handling $T_r + T_b + T_c$, in case of failure T_c is the membership confirmation time.

Table continued.

Protocol used	No. of messages	Time taken
Asynchronous membership algorithm A.	$1_o + (n+1)_s$ b'cast messages. 1 ordinary msg. and $(n+1)$ special msgs.	$2 * T_t$ T_t is the time taken to place a msg. in total order.
Asynchronous membership algorithm B.	2_s i.e. two special b'cast msgs.	$2 * T_t$
NEW_SELF algorithm for diagnosis.	$n * (4t)$ t is the maximum no. of nodes any node tests.	$T_p(n/t+1)$ T_p is the time taken for point to point communication.
EVENT_SELF algorithm for diagnosis.	$n(2+2+2f)$	$n * T_c$ T_c is the time taken by a test cycle.

For each of the above described protocols, these can be seen as following a sequence of the following three stages. The first stage being the transmission of a message. The second stage includes the validation of the message and the confirmation that it has been correctly received. The third stage is to commit the message to the application.

The second stage for different protocols differs depending on the protocol used, and this causes the variation in the complexities for different protocols. Different protocols use different schemes, which may be classified as those using time out after transmission, and those using message ordering schemes to confirm the reception of the message received.

The network systems that are synchronous, i.e. where the clocks at different nodes are within some maximum variation of each other, use the protocols using time out after transmission scheme. This means that all events are time stamped in terms of the clock used, and each event takes a fixed amount of time to complete. For example for a broadcast this time period includes the transmission time, acknowledgment time and retransmission time. If a node fails to respond within the time period that has been specified, it is assumed to have failed.

Certain protocols use the concept of message ordering. This is used where the clocks at different nodes are not synchronized. These protocols use sequence the messages using some protocol as known to all nodes. In these protocols the node constructs an order on the messages before committing them to the application. In such protocols the time needed for a node to commit a message to the application is not bound in terms of units of time taken. Complexity based on actual time taken cannot be determined. Here time taken depends on the amount of time taken to place a message in order. This time is not a fixed in terms of units of time taken, and varies at different nodes and for different messages. In order to determine the complexity of such protocols, we have assumed a time period as an average time period that a message or a group of messages need, before they can be committed to the application.

It can be seen that for all protocols used there has been a trade-off between number of messages used, and time taken to achieve a certain property. This comparison can be made for the case of the two diagnosis algorithms that have been described, i.e. the NEW_SELF algorithm, and the EVENT_SELF algorithm. When a node tests more than one node in a given time period, the number of messages increases but the time taken is reduced by a factor t (approx.), the number of nodes which are tested in parallel. The case where a node communicates and tests a single node in a test period, the number of messages is limited per test period, but time taken to achieve the diagnosis increases. This means that if a node has failed or recovered, then the time it takes for all nodes to know of it's changed status is more. We can hence conclude that for systems where the frequency of failures and recoveries is less, the algorithm requiring fewer messages can be used, whereas the system where there are frequent failures and recoveries, the algorithm with lower latency to achieve diagnosis should be used.

It has been seen that the failure of nodes cause an increase in the number of messages needed. This is the case for all protocols, as on encountering any failure, the nodes communicating with the failed nodes have to communicate with other nodes, in order to achieve the desired property. This is because on failure of a node, some message may have to be retransmitted to other nodes using some other path, as failed node does not respond to messages sent to it. Also as a failed node does not acknowledge a message within some time of it being sent, the message may be retransmitted adding to the number of messages, In the EVENT_SELF diagnosis protocol, on testing a node as faulty, the tester has to test

another node until it encounters a fault-free node, hence increasing the number of messages. In case of the attendance list protocol for membership, it takes n messages to renew the membership in case there is no failure, whereas $(n+1)*(n*d-n+1)+n$ messages are needed to achieve consistency in case of a failure. In case of FTB protocol for reliable broadcast, a maximum of $2n*(n-1)$ messages are needed, but if no failures occur then the number of messages needed is less than $2n*(d-1)$ which is much less than that used in case failures occur.

Chapter 4

Relationship between the Properties

In this section we develop relationship between the different properties. We show that in various situations, a protocol implementing a property can easily be extended to implement another property. We show the relationship between RB and membership and between RB and diagnosis.

4.1 Reliable Broadcast and Membership

Here we show that a protocol satisfying the reliable broadcast property can be used to implement the membership property and vice versa.

4.1.1 Membership using Reliable broadcast

In this section we show that membership can be achieved, given a system that supports reliable broadcast. To achieve membership from reliable broadcast we view membership as a special case of reliable broadcast. Assume that only those messages that contain information regarding either failure or a recovery of a processor are broadcast by a reliable broadcast protocol. This means that a node initiates the reliable broadcast protocol only if it detects the failure or recovery of a node, and by the reliable broadcast property this message is received by all alive nodes.

By restricting the usage of a reliable broadcast protocol in this manner, the membership property can easily be satisfied. At the occurrence of the event the corresponding processor takes appropriate action depending on the type of event and as per the algorithms given for these events. For membership, the initiator and receiver nodes should perform the following

activities :

At the initiator
begin
    initiate_RB(m);
    If $m.info = recovered(x)$ **then**
    begin
        $ML_i = ML_i + \{x\}$;
        $send_i(x, ML_i)$;
    end
    else if $m.info = failed(x)$ **then**
        $ML_i = ML_i - \{x\}$;
end.

In the above module *initiate_RB* initiates the RB protocol for a message m at the initiator. $m.info$ is the information contained in the message. $send_i(x, ML_i)$ is a point to point communication where node i sends the new ML to the recovered node. ML_i is the membership list at initiator node.

At the receiver
begin
    receive_RB_msg(m);
    if $m.info = failed(x)$ **then**
      $ML_r = ML_r - \{x\}$
    else if $m.info = recovered(x)$ **then**
        $ML_r = ML_r + \{x\}$;
end.

In the above module *receive_RB_msg*(m) receives the message m that has been broadcast by the initiator. ML_r is the membership list at receiver node. All other terms are same as in the initiator module.

If a failure or a recovery is detected then the algorithm for the initiator is invoked. The initiator broadcasts the message to all the other alive nodes in the configuration. It adds the message to it's message list. In case of a recovery message the initiator first adds the new node to it's ML. It then sends the new membership list to the recovered process. If recovery of a failed processor is detected at time T it is included in the ML of all members of the configuration at time $T + D$. All messages broadcast from time T to $T + D$ are delivered to the recovered processor by the initiator node. In case of a failure the initiator deletes the failed node from it's ML.

In the event of receiving a message, the algorithm for receivers is invoked. Receiver receives the message and takes appropriate action depending on whether the message is a failure or a recovery message. In case of a failure, the failed node is removed from ML at all the receiver nodes by time $T + D$. In case of a recovery the recovered node is added to ML at all the receiver nodes by time $T + D$.

Claim :If the RB holds true then for the reliable broadcast protocol used for sending messages the above approach achieves one membership protocol.

Proof : The reliable broadcast holds true i.e. any message broadcast at time T will be received by all alive nodes at time $T + D$. We have to show that if the initial condition holds true then using reliable broadcast we can achieve membership i.e., failure or recovery of any node in the configuration if detected, will be known to all alive nodes within a fixed time D. After the transition period i.e. D time period after the RB initiation, the initial condition should still hold true.

At time T initial condition is satisfied assuming stable state of the system. At a failure or recovery, initiator initiates a reliable broadcast, say at time T. All processors receive the message sent at time T by $T + D$. This is guaranteed by reliable broadcast property. Message received by all alive processors is the same. This is true as the system model ensures fault free transmission. All receiver nodes take similar action (all receiver nodes have same protocol). On failure, the failed processor is deleted from the membership list by time $T + D$ at all processors. On recovery the recovered processor is added to the membership list by time $T + D$ at all processors . The above specified arguments ensure that by time $T + D$ status change is registered at all alive nodes. This is equivalent to the membership property i.e.

$$MP \equiv StatusChange(S_1, S_2, i, T) \Rightarrow$$
$$\forall j[RegisterChange(S_1, S_2, ML_j, T + D) \wedge Alive(j, T, T + D)$$

At time $T+D$ the membership lists at all alive nodes are the same if no new status change is recorded i.e. within time D after the first change, and hence the stable state is reached. The claim that, given RB we can achieve membership, is true.

4.1.2 Reliable Broadcast using Membership

In this section we show how a reliable broadcast protocol can be constructed from a protocol that assumes that the membership property holds true. Recall that the membership property states that if any node i sees a change in status at time T, the change in status is registered in the membership list of all alive processors at time $T + D_2$. For membership we classify the nodes into two types:

1. Initiator node : Detects the change in status of any node in the configuration at time T and hence initiates the membership protocol.

2. Other nodes : Register the change in status specified by the initiator node and update their membership lists at time $T + D_2$.

In case of membership protocol the status change is defined as a failure or recovery event. For supporting reliable broadcast we view reliable broadcast as a general case of membership. For this we generalize the existing membership protocol. The definition of status change is modified so that it also includes the event message broadcast. This means that when a node wants to broadcast a message it is viewed as a change in the node's status, and hence the membership protocol is initiated. The actions of the initiator and other nodes are modified and given below so as to satisfy RB.

At the initiator
begin
    if *status_change*(T) **then**
      construct(*status_chg_msg*);
    if *status_chg_msg.status* = *new_msg*(m) **then**
    begin
        status_chg_msg.data = m;
        initiate_membership(*status_chg_msg*);
    end;
end.

In the above module *status_change*(T) is true if a change in status has been detected by a node at time T. *construct*(*status_chg_msg*) constructs a message containing the information of the change in status that has occurred. *new_msg*(m) states that a message m is ready for broadcast at the node. *initiate_membership*(*status_chg_msg*) initiates the membership protocol.

```
At the other nodes
begin
        register_membership(status_chg_msg);
        if status_chg_msg.status = new_msg(m) then
        begin
                m = status_chg_msg.data;
                message_list = message_list + m;
        end
end.
```

In the above module *register_membership(status_chg_msg)* means that the new membership change as given in the *status_chg_msg* is registered at the node. *message_list* is a list of messages received by a node. All other terms are same as in the initiator module.

When a status change is detected by a node it becomes the initiator node. The event causing a status change may be a failure, a recovery or a new message which has to be broadcast. The node that detects the status change constructs a message i.e. the *status_chg_msg* in order to send this to the other nodes to inform about the status change. For our purpose i.e. to achieve broadcast we assume a data field in every message. If the new event is a new message then along with the information of status change the new message is also sent in the data field of the message. The new message is added to the message list and the membership protocol is invoked. By membership property the change in membership as specified in the *status_chg_msg* should be recorded at all other alive nodes within some bounded time. This means that all the nodes that obtain the information of status change also get the new message.

The other nodes register the status change at a bounded time (say D_2) after it is recorded at the initiator(at time T). At these nodes the event is recorded in the event list. If the new event is a new message, then it is accepted along with the status information and is added to the message list.

Claim : If the membership holds true, then for the membership protocol used for maintaining consistent information about failure and recovery at all alive processors, we can achieve a reliable broadcast protocol by using the above approach.

Proof : Membership holds true i.e. any status change detected at time T will be registered by all alive nodes by time $T + D_2$. All alive processors contain a list of messages since the time they are alive. If the processors do not encounter any failures or recoveries then

message list at all processors which joined the configuration at the same time should be same. Initially all processors have their lists empty as no messages have been broadcast.

At a status change at time T, the initiator node executes the membership protocol which also includes the event of sending a new message. All alive processors register the status change detected at time T, by time $T + D_2$. This is guaranteed by the membership property. All nodes take similar action (all receiver nodes have same protocol) on receiving the message as message received by all nodes will be same. If the new event is a new message then this message is added to the message list at all nodes by time $T + D_2$ as specified in the module. The above specified arguments ensure that by time $T + D_2$ the new message is recorded at all alive nodes. This is equivalent to the reliable broadcast property where a reliable broadcast takes a fixed time D_2 to complete i.e.

$$RB \equiv Sent(m, T, i) \Rightarrow \forall j[Received(m, T + D_2, j) \wedge Alive(j, T, T + D_2)]$$

At time $T + D_2$ the message lists at all alive nodes are same. This means that given membership protocol, RB can be achieved.

□

4.2 Reliable Broadcast and Diagnosis

In the following section we show how a protocol satisfying the reliable broadcast property can be used to implement the diagnosis property and vice versa.

4.2.1 Diagnosis using Reliable broadcast

Here we show that diagnosis can be achieved given a system that supports reliable broadcast. To achieve diagnosis from reliable broadcast we view diagnosis as a special case of reliable broadcast. Only those messages that contain information regarding either failure or a recovery of a processor are broadcast by the reliable broadcast protocol. This means that a node initiates the reliable broadcast protocol only if it detects the failure of a node or recovery of a node. On detecting a change in status of any node x, a node i will initiate a RB. All other nodes will receive the RB message and take appropriate action as specified by the module at the receiver. The existing protocol for the initiator and receiver nodes should be modified to include the following routines :

At the initiator

```
begin
        initiate_RB(m);
        If m.info = recovered(x) then
        begin
                FS_i[x] = 0;
                send_i(x, FS_i);
        end
        else if m.info = failed(x) then
                FS_i[x] = 1;
end.
```

In the above module, $initiate\_RB(m)$ initiates the RB protocol for a message m at the initiator. $m.info$ is the information contained in the message. FS_i is the data structure at node i and it contains the fault situation of all nodes. $send_i(x, FS_i)$ is a point to point communication where node i sends the new fault situation to the recovered node.

```
At the receiver
begin
        receive_RB_msg(m);
        if m.info = failed(x) then
          FS_r[x] = 1
        else if m.info = recovered(x) then
                FS_r[x] = 0;
end.
```

Here $receive\_RB\_msg(m)$ broadcasts the message m. FS_r is the data structure at all receiver nodes containing the fault situation of all nodes. $FS_r[x]$ is the fault situation of the node x. All other terms are the same as in the initiator module.

If a failure or recovery is detected, the initiator initiates a reliable broadcast in order to send the information of status change to all processors. It then checks the information to see if it is a failure or a recovery. On detecting a recovery it updates the value of the recovered processor in it's FS and sends this new FS to the recovered processor. On detecting a failure it updates the value of the failed processor in it's FS.

On receiving the broadcast message m, the receiver checks the information to see if it is a failure or a recovery. On detecting a recovery, it updates the value of the recovered processor in it's FS. On detecting a failure, it updates the value of the failed processor in it's FS.

Claim :If the RB holds true, then for the reliable broadcast protocol used for sending

messages the above approach achieves one diagnosis protocol.

Proof : The reliable broadcast holds true i.e. any message broadcast at time T will be received by all alive nodes by time $T + D$.

Initially the FS at all nodes is same and contains a 0 value for all nodes i in $FS[i]$ specifying that all nodes are alive. When a failure or a recovery is detected, a message to inform all nodes is broadcast, say at time T. All processors receive the message sent at time T by time $T + D$. This is guaranteed by reliable broadcast property. Messages received by all alive processors will be the same. This is true as the system model ensures fault free transmission. All nodes on receiving the message update their FS. The above specified arguments ensure that by time $T + D$ fault situation is registered at all alive nodes. This is equivalent to the diagnosis property i.e.

$$DP \equiv FS_g(T) \Rightarrow \forall i, j[equal(FS_i, FS_j, T + D_3) \wedge Alive(i, T, T + D) \wedge Alive(j, T, T + D)]$$

By time $T + D$ the FS at all nodes is the same. This means that given the RB protocol, diagnosis can be achieved.

□

4.2.2 Reliable Broadcast using Diagnosis

In the following section we show how the diagnosis protocol is used to achieve reliable broadcast. To attain RB we place certain restrictions on the existing diagnosis protocol and modify it accordingly. We assume that on being tested, a node tests positive for a status change if it has a message to broadcast. The test results(TR) reflect one of the three conditions that cause a status change i.e. a failed node, an alive node or a new message to broadcast. If the status change is due to a new message, the message is also sent along with the test result. Diagnosis is performed on a bit vector which contains the test results, after receiving the test information sent by other nodes. We assume that instead of a bit vector each node collects the information in form of a $n * (m + 2)$ matrix where m specifies the maximum length of a message. We also modify the diagnosis protocol such that instead of performing the diagnosis on bit vector, the diagnosis is performed on the modified test result.

In the modified test result the first bit of each row specifies if a node is failed or alive as determined by the test. Bit 2 specifies if a node has a message to broadcast, and if this bit

is 1 the remaining m bits carry the message. This means that the first two bits can specify any of the three status change conditions i.e. failed node, alive node and a new message to broadcast.

- 00 : It specifies that a node is alive but has no message to broadcast.
- 01 : It means that a node is alive and has a new message to broadcast.
- 10,11 : These two specify a failed node. As we are assuming failed-stop processors, there will be no message to broadcast.

Initially all nodes have $TR[i,2]$ as 0 for all i. This means that the nodes have no message to broadcast. When a node i has a message to broadcast it changes the $TR[i,2]$ to 1 and initiates the diagnosis protocol. When a tester node tests node i, it finds that the status of a node has changed. This new test result is made known at all nodes, and along with it the new message is also sent. It is essential that each node has test results for all other nodes in order to achieve diagnosis. If a node sends a message at time T, then by time $T + D_3$ the message will be received by all nodes that are alive where D_3 is the time taken to achieve diagnosis after any new change in status of any node. At the end of every test period the $TR[i,2]$ is set to 0 for all i. In the modified protocol, the information that was being sent earlier is still being conveyed and hence the diagnosis protocol can still be used as before to achieve diagnosis. The only modification done for our purpose is to include the information of a new message, and if the message is present, to append it to the test result.

At the tester node, on performing the test, the information to determine the fault situation is obtained from different nodes. The fault situation is derived from this information that is collected. If second bit of any row is 1, it means that the node has sent a new message. This message is read into *new_msg* and added to the message list. At the node being tested, if node i wants to broadcast a message at time T, it sends this information as a part of the test result along with the message.

Claim : If the diagnosis holds true, then the diagnosis protocol that is used to provide a consistent information about the fault condition of all the nodes of a network can achieve reliable broadcast using the above approach.

Proof : Diagnosis holds true, i.e. fault situation of all nodes will be the same at all nodes, every D_3 time units after a change is detected.

In diagnosis, a change in fault condition of a node is known to all nodes within a fixed time after it's detection. This change is made known by the test results obtained. Instead of the status of nodes i.e. failed or alive, fault condition also depends on whether a node wants to broadcast a message. The message is transmitted along with the test results. As test results for all nodes reach all alive nodes, the message will also be received by all alive nodes. This is equivalent to the RB property i.e.

$$RB \equiv Sent(m,T,i) \Rightarrow \forall j[Received(m,T+D_3,j) \wedge Alive(j,T,T+D_3)]$$

By time $T + D_3$ the message lists at all alive nodes are same. This means that given diagnosis protocol, reliable broadcast is achieved within a fixed time after a new message is ready for broadcast.

□

Chapter 5

Conversion Examples

In this chapter we have used one of the existing protocols for each of the properties discussed, and "converted" it to implement another property, using the methods discussed in the previous chapter.

5.1 Membership using the FTB protocol

In this example we see if the FTB protocol for reliable broadcast can be used to achieve membership by using the modifications made. The properties of the membership protocol thus obtained will also be discussed.

The RB protocol has been modified to assume that RB takes place only if a message carries a status change information. As specified in the protocol for FTB, the failure of a node is detected by some other alive node without performing any special tests. Similarly a recovered node is detected by some node when the recovered node sends a message to it regarding it's recovery. As specified, in order to achieve membership, the node that detects the status change, initiates a reliable broadcast of message m. The information contained in the RB message is that of a status change. As per the FTB protocol for reliable broadcast, the message m is sent to the root node, for the reliable broadcast. All alive nodes receive the message that has been broadcast within a finite time after it had been broadcast. This is ensured by the RB protocol used.

The initiator node performs the following functions depending on the steps specified in the modified initiator module. After initiating the reliable broadcast, it checks the information contained in the message. If $m.info$ i.e. the information field of the message contains a recovery information, the recovered node's identifier is included in the membership list. In this case, the parent of the recovered node sends all the messages that the node may

have missed or is likely to miss. In the modified protocol, this message is replaced by a membership list. The node that detected the recovered node sends a list of all the nodes that are members of the group and are presently alive, after including the recovered node in the membership list. If $m.info$ contains a message regarding failure of a node, the identifier of the failed node is removed from the membership list.

At the receiver nodes, all the nodes receiving the RB message check the $m.info$ field for the information contained in the message that has been broadcast. If $m.info$ i.e. the information field of the message contains a recovery information, the recovered node's identifier is included in the membership list. If $m.info$ contains a message regarding failure of a node, the identifier of the failed node is removed from the membership list.

It can be seen that, all alive nodes in the system register the membership status change, a finite time units after it has been detected. This is equivalent to what a membership protocol aims at achieving. It can also be seen that, no special messages have been used in order to modify the protocol and that the number of messages used is the same as that used for achieving reliable broadcast. The time taken to achieve the membership is also approximately same as that for achieving reliable broadcast, as major part of the time comprises of the time taken for communication which is same in both cases. The properties of the membership protocol that results from the FTB protocol are :

- Accuracy and Liveness : The recovery detection of a node will be accurate, as the recovered node itself prompts another node of it's recovery, but it may not be live. The failure detection is live i.e. any failed node will eventually be detected as failed, but failure detection may not be accurate.

- Agreement : All alive nodes in the system agree on any status change, a fixed time after the change has been detected i.e. the membership lists at all alive nodes is same in stable state.

- Confidence : In this case the degree of confidence in the status change(failure) is low. This is because in the reliable broadcast protocol, there is no voting involved among the members at the time of detecting the failure and the decision is based in the suspicion of a single node.

- Ordering : In this protocol, a total order in messages is achieved. This is because, all messages are first sent to the root node, which then places the messages from all

different processors in a sequence which is common for messages.

5.2 Diagnosis using FTB protocol

In the FTB protocol used, the initiator node initiates a reliable broadcast only if it detects a failure or recovery of a node. In the diagnosis algorithms described, it has been seen that in order to detect the change in status of a node, nodes are required to test other nodes. In the RB protocol being used, the detection of failure and recovery are not done by any special tests. The status change is detected inherently by the FTB protocol.

In the FTB protocol used, the action taken at the initiator node and at the receiver nodes as specified by the modified protocols is as follows. As soon as a node detects a change in status of a node x, it initiates a reliable broadcast. The initiator sends the message containing the status change information to the root node at which all broadcasts are initiated. The initiator then checks the $m.info$ i.e. the information contained in the message. If it is a recovery of a node x, then the fault situation value of the node x i.e. $FS_i[x]$ is set to 0, which means that the node is alive. The set containing the fault situation of all the nodes i.e. FS is transmitted to the recovered node, by the initiator. If the information contained in the message states that a node x has failed, the value $FS_i[x]$ is set to 1, stating that the node x is faulty.

At the receiver nodes, all nodes receive the message and check the information contained in the message. Depending on whether it is a failure or a recovery of a node x, the $FS_r[x]$ value is set to 1 or 0. As the FTB protocol is a reliable broadcast protocol, it ensures that any message broadcast by an alive node will reach all other nodes within a fixed time after the broadcast. This means that after the status change is detected, the fault situation at all alive nodes is updated within a finite time and reflects the change in the fault situation of the system. This is equivalent to the results obtained if a system diagnosis is performed.

As seen in the example, the protocol does not need any special message other than those needed for the FTB. Time taken to achieve the diagnosis is also the same as that needed to achieve the reliable broadcast. This means that the FTB protocol can successfully achieve diagnosis without requiring any additional resources in terms of messages, tests and time.

5.3 Reliable Broadcast using Partial Order Membership Protocol

In the following section, the partial order protocol for membership is being used to achieve a reliable broadcast protocol. The modifications that are to be done to an existing membership protocol in order to achieve reliable broadcast have been discussed earlier. As we have seen, the status change as per the new definition, also includes the event of a new message to be broadcast. In the modified protocol for membership, the protocol is initiated in case of a failure detection, detection of recovery of a processor, and in case when a node has a message to broadcast. The functions to be performed by the initiator and the other nodes are described as follows.

When a node has a message to broadcast, it is detected by the detection protocol. The detection protocol constructs the status change message that includes the message to be broadcast in it's data field, and the status field of which depicts the type of change. Then the membership protocol is initiated with *status_chg_msg* as it's parameter. In the membership protocol by partial order, the *status_chg_msg* is first send to the conversation abstraction. All alive nodes receive the message.

At the other nodes, all nodes on receiving the message check the status change that has taken place. If the status change is due to a new message, then the message is copied from the data field of the *status_chg_msg* to a buffer from where it is copied to the message list. The messages are committed to the application as soon as they become stable. As soon as a node receives a message, it checks the message and returns an ack to the conversation regarding the message contained. When an ack from all the nodes is received, the message becomes stable. This confirms that the message has reached all the nodes. In order to achieve the reliable broadcast, no special messages have been used other than those used in the membership protocol. This is equivalent to a reliable broadcast protocol.

5.4 Diagnosis using Partial Order Membership Protocol

In order to achieve a diagnosis protocol from the membership protocol using partial order, the steps are given below.

- Status change is suspected by the detection protocol and it sends a suspect message to the conversation. If the suspect message is that of failure of a node, all nodes stop

receiving any new messages from that node.

- If some node detects that the suspected status change is false, it sends a nack message to the conversation and the original status of the node is restored.
- On receiving an ack from all the nodes, the suspected status change is confirmed, and membership list at the node is updated. If the confirmation is that of a recovery of a node, all nodes start accepting messages from the recovered node.
- In order to achieve diagnosis, as soon as a status change is confirmed at a node, the fault situation of the node in the set containing the fault situation of all nodes, is updated. In case of recovery of one or more nodes, the fault situation values in the set is changed to '0', and in case of failures, the values are set to '1'.

As the membership protocol ensures that the membership change in the system is reflected at all alive nodes within a fixed time after the change has been detected, the values in the FS at each alive node also get modified. Some difference in the diagnosis protocol thus obtained has been observed from those that have been originally described. In the diagnosis protocols, the fault situation of a system at a given time T is diagnosed within a fixed time delay after T at all nodes. In the diagnosis protocol obtained above, the diagnosis is not obtained at all the nodes at a fixed time delay after initiation of the membership protocol, but it is obtained at the same logical time i.e. at the same point in the order of events occurring. This is due to the fact that unlike the diagnosis protocol's system model, the nodes of the underlying system model in the membership protocol used are not fully synchronized. But whatever method used for detection, the final results obtained are equivalent to those of a diagnosis protocol. No special messages or tests are required to achieve the diagnosis from the membership protocol.

5.5 Reliable Broadcast using Event_Self Diagnosis Protocol

In the Event_Self diagnosis protocol, each node possesses the capability to test at least one other fault-free node. In order to obtain a RB protocol using the Event_Self protocol, it is used along with the modifications as discussed earlier. In case of the modified protocol, a status change is said to have occurred in case of a change in status of a node from faulty to fault-free, from fault-free to faulty and in case when the node has a new message to

broadcast. The following steps are followed at all the nodes.

The tester node say x tests another node y, in order to determine the status of the node. If the node is faulty, node x will test another node and this continues until node x finds a fault- free node. For all nodes tested faulty, node x will store the test result as 10 stating that the node is faulty. In case where the test result suggests that the tested node has a new message to broadcast, the message to be sent is included in the test result. The tester node receives the test results of the node tested, and other information collected by the node being tested, and stores it in a temporary buffer. Tester node again tests the previously tested node to confirm that it's status has not changed. If the status remains including the message to be transmitted, then the test results(including the message) and other information gathered from the tested node is assumed to be valid. If the test results of the tester node x contains a new message, the message is stored in the bits 2 to m+1 in the bit vector of the node tested , that stores the diagnostic information to be forwarded to the tester of node x. The bits 0 and 1 of the bit vector are set to 01. The diagnostic information stored in the bit vector of a node is forwarded to it's tester node, and this continues until all nodes receive the information for all other nodes. After completing one cycle, which started at time T, all nodes have complete diagnostic information. This includes the messages that had been sent by all the nodes at time T. If the completion of a cycle takes D_3 amount of time units, then by time $T + D_3$ all messages that were to be broadcast at time T are received at all nodes. This is equivalent to a reliable broadcast.

5.6 Membership using Event_Self Diagnosis Protocol

For the Event_Self diagnosis protocol used, the fault situation of all nodes at time T is known to all nodes at time $T + D_3$. The fault situation of all nodes is represented as a set of binary values $\{s_1, s_2, ..., s_n\}$, where s_i is 0 if node i is alive, and 1 if node i is faulty. To achieve membership from this set is very simple. A membership list is available at all nodes which contains the identifiers of all those nodes that are alive at that time. In order to achieve membership, the membership list at a node is updated as soon as a new diagnostic information is available and is validated. The membership list is updated by adding or removing the identifier of the recovered or failed node respectively from the previous membership list. Since all alive nodes come to know of the status of all other nodes at time T by time $T + D_3$, the membership lists at time $T + D_3$ contain the list of

all nodes that are alive at time T. The restriction in the diagnosis protocol is that, in the diagnosis protocol the tests for checking the status of nodes are performed at fixed intervals of time. This means that the node failures and recoveries in this case will be detected at those time intervals only. Any node failure or recovery occurring in between these intervals will not be immediately detected, but only when the next test is performed. Also the basic assumption of the diagnosis protocol that, no node can fail and recover from a failure in an undetected manner in the duration between the two tests by a node, should hold true. If this assumption is not true the failure or recovery may go unnoticed. The membership protocol thus obtained does not require any special messages other than those used by the diagnosis protocol. The membership properties that are fulfilled by the membership protocol thus obtained are stated below.

- Accuracy and Liveness : In this case each node of the underlying diagnosis protocol tests at least one other correct node. All the nodes detected as failed are also tested in the next test cycle for recovery. As all nodes are tested and test results for each of these nodes is obtained in every cycle, the membership protocol thus obtained will possess both accuracy and liveness for failure and recovery detection.

- Agreement : As all nodes use the same protocol for obtaining a diagnosis, and the information used to achieve the diagnosis is same at all nodes, all nodes agree on the diagnosed fault situation and hence on the membership. The membership lists at all alive nodes is same after every test cycle is complete.

- Partition handling : The diagnosis protocol assumes that the network does not partition on any number of failures. Hence no partition handling is required.

- Confidence : In this case each node individually takes the decision based on the test results obtained, which is dependent on the diagnosis protocol used. Hence in this case the degree of confidence will depend on the diagnosis protocol used.

- Ordering : As the system modes assumes that all nodes are synchronized, the messages are time stamped by each node and hence a FIFO order on messages is maintained.

Chapter 6

Conclusions

Distributed systems are prone to situations where there is inconsistency in information and lack of coordination among different processors. Different properties for distributed systems, namely reliable broadcast, membership, and diagnosis have been studied. In a system, where faulty and fault-free processors co- exist, and in presence of continued failures and recoveries, the protocols for these three properties aim at achieving consistency at each alive processor. Diagnosis attains by each node testing other nodes, identifying faulty from fault-free nodes, and each node independently reaching a decision regarding the fault situation of the system. In membership, the failures and recoveries of nodes are inherently diagnosed, without any special testing.

After studying the various protocols, each property has been defined formally so as to understand it's significance in the given environment. As presently defined, each of these protocols can fulfill only one of the properties, for which it has been specifically developed. It was however seen that the basic aim of each protocol is to achieve some consistency among the different processors. With this observation, these protocols have been further compared and an effort has been made to determine if all three properties can be attained using just one of these protocols. The study has revealed that although achieving different properties, the protocols are based on similar principles, and aim at achieving a common goal, though at a higher level, performing different functions. Similarity between these has been proved by using protocols for each of these properties to extend them to successfully achieve the other two.

It has been shown that the reliable broadcast protocol can be used to achieve membership and diagnosis. Also, it has been determined that the membership and the diagnosis protocols can be used to successfully perform reliable broadcast. This also concludes that

the membership protocol can be used for achieving diagnosis. Membership would be achievable for a given diagnosis protocol. Thus it has been seen that all these protocols can be used interchangeably. The three consistency properties i.e. reliable broadcast, membership and diagnosis which have until now been studied in isolation are basically similar and hence for future purposes should be studied together.

The various existing protocols for each of these properties have been described. A complexity analysis for these protocols has been done on the basis of number of messages needed by each, and time taken by each protocol to achieve the desired property. It has been seen that the time taken and number of messages needed increase for all protocols, in case of recurring failures. Similarly it has been seen that there is a trade-off in the number of messages needed and time taken for most of the protocols. Also the number of messages and the time taken vary with the degree of the nodes in the network i.e. they are dependent on the configuration of the communication network.

We can hence conclude that for networks where time is an important parameter, compromise should be done on the number of messages used , and network topology should support higher traffic. If in a systems failure rate is low and time delay does not affect the applications to a great extent, then the traffic in the network can be reduced although time taken to achieve a property may be more.

We have assumed that the network always remains connected. Future work is needed to study the effect of partitions.

The motivation for comparing the properties that have been compared was derived from the fact that all these properties aim at achieving a consensus in a distributed system. Certain other ways of achieving consensus in distributed systems are available. A study in order to compare these with the ones already discussed can be done. Furthermore, in light of the similarities that we have shown, a few higher level primitives can be identified for a distributed system, which can be used to build a protocol for any consistency service. Further work is needed to identify such primitives and the various services that can be built using the primitives.

Bibliography

[1] R. Bianchini, K. Goodwin, D. S. Nydick, " Practical Application and Implementation of Distributed System-Level Diagnosis Theory", IEEE 1990.

[2] R. Bianchini, R. Buskens, "An Adaptive Distributed System-Level Diagnosis Algorithm and Its Implementation", IEEE 1991.

[3] R. P. Bianchini, R. W. Buskens, " Implementation of On-Line Distributed System-Level Diagnosis Theory", *IEEE Transactions on Computers*, Vol 41, No. 5, May 1992.

[4] K. P. Birman, T. A. Joseph, " Reliable communications in the Presence of Failures" *ACM Transactions on Computer Systems* Vol. 5, No. 1, August 1987.

[5] Jo-Mei Chang, N.F. Maxemchuck, "Reliable Broadcast Protocols" *ACM Transactions on Computing Systems* Vol. 2, No. 3, August 1984.

[6] G. Coulouris, J. Dollimore, T. Kindberg, " Distributed Systems Concepts and Design", *Addison Wesley Publishing Company* Second Edition.

[7] Flaviu Cristian, " Agreeing on who is Present and who is Absent in a Synchronous Distributed System", IEEE 1988.

[8] S. H. Hosseine, J. G. Kuhl, S. M. Reddy, " A Diagnosis Algorithm for Distributed Computing Systems with Dynamic Failure and Repair", *IEEE Transactions on Computers*, Vol. C-33, No. 3, March 1984.

[9] M. A. Hiltunen, R. D. Schlichting, " Properties of Membership Services" *Proceedings of the 2nd International Symposium on Autonomous Decentralized Systems*, Phoenix, AZ, USA, April 1995.

[10] S. E. Kreutzer, S. L. Hakimi, " System-Level Fault Diagnosis: A Survey", *Microprocessing and Microprogramming*, 20 (1987) 323-330

[11] S. Mishra, L. L. Peterson, R. D. Schlichting, "A Membership Protocol Based on Partial Order",

[12] L. E. Moser, P. M. Melliar-Smith, Vivek Agrawala, " Membership Algorithms for Asynchronous Distributed Systems", IEEE 1991.

[13] L. L. Peterson, N. C. Buchholz, R. D. Schlichting, " Preserving and Using Context Information in Interprocess Communication", *acm Transactions on Computer Systems*, Vol 7, No. 3, August 1989.

[14] A. M. Ricciardi, K. P. Birman, " Using Process Groups to Implement Failure Detection in Asynchronous Environments.", ACM 1991.

[15] R. D. Schlichting, F. B. Schneider, " Fail-Stop Processors: An Approach to Designing Faulr-Tolerant Computing Systems", *ACM Transactions on Computing Systems* Vol. 1, No. 3, August 1983.

[16] F. B. Schneider, D. Gries, R. D. Schlichting, "Fault Tolerant Broadcast", *Science of Computer Programming 4*, 1984, pp 1-15.

[17] P. M. Melliar-Smith, L. E. Moser, Vivek Agrawala, " Broadcast Protocols for Distributed Systems", *IEEE Transactions on Parallel and Distributed Systems*, Vol. 1, No. 1, January 1990.